Tutorial: A scrolling shooter

Last changed: April 18, 2003
Uses: version 5.0, advanced mode
Level: Beginner

"Scrolling shooters" sind eine sehr beliebte Art von "arcade action" Spielen, welche ziemlich einfach zu entwickeln sind, mit einem Softwarepaket wie dem "Game Maker".
In diesem Tutorium werden wir solch ein Spiel erstellen und während des Entstehungsprozeßes werden wir einige Aspekte vom "Game Maker", wie zum Beispiel den Gebrauch von Variablen, kennenlernen.
In einem "scrolling shooter" kontrolliert der Spieler ein Objekt, wie ein Flugzeug, Raumschiff, oder ein Auto, welches sich über einen "scrolling background" ("sich verschiebenden Hintergrund") bewegt. Hindernisse tauchen auf dem Hintergrund auf, die man meiden sollte und Gegner erscheinen, die man abschießen muß. Oftmals kommen Bonusobjekte vor, die man aufsammeln muß, um zusätzliche Vorteile zu erlangen. Im Verlauf des Spiels tauchen immer mehr und andersartige Gegner auf und sie werden immer stärker, was ein Überleben nach und nach erschwert.
In diesem Tutoium werden wir einen "scrolling shooter" namens 1945 entwickeln, in diesem fliegt der Spieler ein Flugzeug über dem Meer und feindliche Flugzeuge versuchen den Spieler zu zerstören. Wir werden Themen behandeln, wie beispielsweise den Effekt einer Bewegungsillusion durch einen "scrollenden" Hintergrund, wie man das Flugzeug steuert, wie man Gegner und Projektile erstellt und wie man den Punktestand, die Anzahl der Leben und den Schaden am Flugzeug verwaltet. Aber zuallererst beschäftigen wir uns mit einem sehr wichtigen Aspekt des Game Makers, der die Möglichkeiten beträchtlich erweitert: Der Gebrauch von Variablen.

Variablen und Eigenschaften

Bevor wir jetzt ein "scrolling shooter" Spiel erstellen, müssen wir uns mit einem Konzept des Game Maker näher befassen: Dem Gebrauch von Variablen. Dieses sehr einfache Konzept birgt einen sehr mächtigen Mechanismus in sich, um das "game play" interessanter zu gestalten. Was ist nun so eine Variable?
Betrachten wir sie als eine Eigenschaft einer Instanz eines Objektes. Wie Du wissen solltest, gibt es bestimmte Eigenschaften, die wir angeben können, wenn wir ein Objekt definieren. Beispielsweise können wir festlegen, ob es sichtbar oder ob es "solid" ist. Weiter gibt es noch eine Anzahl von Aktionen (actions), die bestimmte Eigenschaften verändern. Zum Beispiel gibt es eine Aktion, die die Position oder die Geschwindigkeit der Instanz verändert. Jede Instanz besitzt eine Anzahl dieser Eigenschaften und es gibt zudem noch eine Zahl von globalen Eigenschaften, wie zum Beispiel den Punktestand, der keiner einzelnen Instanz zugeordnet wird. Alle Eigenschaften werden in sogenannten Variablen gespeichert, welche eine eigene(n) Bezeichnung(Namen) bekommen. Hier sind ein paar Eigenschaften/Variablen, die jede Instanz besitzt:

x die x-Koordinate der Instanz
y die y-Koordinate der Instanz
hspeed die horizontale Geschwindigkeit (in pixels pro step)
vspeed die vertikale Geschwindigkeit (in pixels pro step)
direction die momentane Richtung der Bewegung in Grad (0-360; 0 ist horizontal nach rechts)

speed die aktuelle Geschwindigkeit in Bewegungsrichtung
visible gibt an, ob das Objekt sichtbar(1) ist oder unsichtbar (0)
solid gibt an, ob das Objekt "solid"(1) oder nicht "solid" (0) ist (solid~massiv/durchlässig)

Und es gibt einige globale Variablen:

score der aktuelle Wert des Punktestandes
lives die momentane Anzahl an Leben
mouse_x x-position der Maus
mouse_y y-position der Maus
room_speed aktuelle Geschwindigkeit des Raumes (in steps pro Sekunde)
room_caption Titel, der in der Titelleiste des Fensters angezeigt wird
room_width Breite des Raumes in Pixel
room_height Höhe des Raumes in Pixel

Es gibt viele weitere Variablen, sowohl lokal für die Instanzen, alsauch global. Sie alle werden in der Dokumentation/Help aufgeführt. Es gibt Aktionen, die den Wert bestimmter Variablen manipulieren aber - wie wir noch sehen werden - kannst Du sie auch direkt verändern. Was gibt es besseres, Du kannst Deine eigenen Variaben definieren und sie genausogut verwenden. Zum Beispiel, wie wir weiter unten noch sehen werden, wollen wir unser Raumschiff nur jeden 5ten "step" des Spieles einmal feuern lassen können. Folglich braucht unser Raumschiff eine Eigenschaft, die angibt, ob geschossen werden darf. Wir verwenden eine Variable, die wir "can_shoot" nennen, um diese Eigenschaft zu realisieren. (Ein Name/Bezeichner einer Variablen darf nur aus Buchstaben und dem Unterstrich gebildet werden. Bei Variablennamen wird Groß-/Kleinschreibung unterschieden, somit ist "Can_Shoot" nicht diesselbe Variable, wie "can_shoot".)
Im "creation event" des Raumschiffes setzen wir diese Variable auf 1 (wir verwenden immer die eins, um anzugeben, daß etwas wahr ist). Wenn der Spieler jetzt schießen möchte, überprüfen wir den Wert dieser Variablen, um festzustellen, ob geschossen werden darf.
Wann immer ein Schuß abgefeuert wurde, setzen wir den Wert auf 0 (um anzugeben, daß etwas falsch ist). Anschließed verwenden wir ein "timer event", um damit den Wert nach fünf "steps" wieder auf eins zu setzen. Weiter unten werden wir das detaillierter beschreiben.
Auf ähnliche Weise können wir Variablen verwenden, um anzugeben, ob das Raumschiff einen aktivierten Schild besitzt, ob es über spezielle Waffenverbesserungen verfügt, usw.
Es gibt zwei wichtige "actions", um Variablen direkt zu behandeln:

Set the value of a variable (Setze den Wert einer Variablen)

Du findest diese "action" unter der Registerkarte "**Code**". Mit dieser Aktion kannst Du den Wert einer gegebenen Variable verändern. Dies kann eine der vorgegebenen (built-in) Variablen sein oder eine von Dir definierte Variable.
Du legst den Namen und den neuen Wert der Variablen fest. Wenn Du einen Haken in die "**Relative box**" (ganz unten das Kästchen, wo "Relative" neben steht) machst, wir der Wert zum aktuellen Wert der Variablen hinzuaddiert. Beachte bitte, daß dies nur geschehen kann, wenn der Variablen vorher ein Wert zugewiesen wurde! Anstatt nur einen einfachen Wert der Variablen zuzuweisen, kann auch ein Ausdruck angegeben werden. Zum Beispiel kannst Du den Punktestand verdoppeln, indem Du den Wert der Variablen "score" auf den Wert "2*score" setzt.

If a variable has a value (Wenn eine Variable einen Wert erreicht)

Du findest diese "action" unter der Registerkarte "**Code**". Mit dieser Aktion kannst Du Prüfen, ob eine Variable einen bestimmten Wert hat. Wenn der Wert der Variablen dem abgefragten entspricht, wird die Aussage als wahr gewertet und die nächste Aktion oder der nächste Block von Aktionen wird ausgeführt. Wenn der Wert nicht gleich ist, wird die nachstehende Aktion oder der nachfolgende Block von Aktionen nicht ausgeführt. Du kannst auch festlegen, daß überprüft werden soll, ob der Wert der Variablen größer oder kleiner als der angegebene wert ist. Du bist nicht auf Variablen beschränkt, sondern Du kannst einen beliebigen Ausdruck zum Prüfen verwenden. Weiter unten werden wir eine Anzahl von Anwendungsbeispielen dieser Aktionen sehen.

Da gibt es noch eine Sache, die Du über Variablen wissen solltest. Wie schon erwähnt gibt es lokale Variablen, die zu einer Instanz "gehören" und es gibt globale Variablen. Wenn Du Deine eigenen Variablen benutzt, sind es immer lokale Variablen, die nur für die Instanz existieren, in deren "actions" Du sie verwendest. Wenn Du eigene globale Variablen verwenden möchtest, mußt Du sie mit dem Präfix "global" und einem Punkt versehen. So kannst Du beispielsweise eine Variable global.bonus benutzen, um die Anzahl der vom Spieler gesammelten Bonuspunkte anzugeben. Achte immer darauf, daß Du Variablennamen vergibst, die nicht schon existieren und sich von vorhandenen Namen für sprites, sounds, usw. unterscheiden. Eine Möglichkeit ist, Deine Variablen immer mit var_ beginnen zu lasssen.

1945

Laß uns nun einen Blick auf das Spiel werfen, welches wir erstellen wollen. Bevor wir mit dem Erstellen eines Spieles anfangen, müssen wir ein Designdokument schreiben. Da das Spiel 1945, daß wir erstellen wollen, ziemlich kompliziert ist, würde ein komplettes Designdokument einige Seiten umfassen. Damit dieses Tutorium nicht zu lang wird, gibt es nur eine kurze Beschreibung. Das Spiel sieht in etwa so aus

1945 design document

Beschreibung

In diesemSpiel fliegst Du ein Flugzeug über dem Meer. Du begegnest einer wachsenden Zahl von feindlichen Flugzeugen, die versuchen Dich zu zerstören. Du solltest ihnen ausweichen oder sie abschießen. Das Ziel ist, solange als möglich am Leben zu bleiben und dabei soviele Feindflieger wie möglich zu eliminieren.

Spielobjekte

Der Hintergrund besteht aus einer scrollenden See mit ein paar Inseln. Das Spielerflugzeug kann über das Meer fliegen. Du kannst Kugeln abschießen, um Feindflugzeuge zu vernichten. Es gibt vier Typen von gegnerischen Flugzeugen:
Ein Flugzeug das Dir begegnet und das Du zerstören solltest.
Ein Flugzeug, das Kugeln nach unten abfeuert.
Ein Flugzeug, das Kugeln auf das Spielerflugzeug abfeuert und ein schnelles Feindflugzeug, daß von hinten anfliegt statt von vorne.

Sounds

Es gibt ein paar Explosionsgeräusche und ein wenig Hintergrundmusik.

Steuerung

Der Spieler steuert das Spiel mit den Pfeiltasten. Mit der Leertaste wird eine Kugel abgefeuert. Nur alle fünf steps (Schritte) kann eine Kugel ausgelöst werden.

Spielverlauf

Der Spieler befindet sich sofort im Spiel. Der Spieler hat drei Leben. Sind alle Leben verwirkt wird eine Highscore-Tabelle angezeigt. Mit der F1-Taste wird eine knappe Spielerklärung angezeigt. Drücken der Esc-Taste beendet das Spiel.

Levels

Es gibt nur einen Level aber mehr und mehr Flugzeuge tauchen auf: zuerst nur die einfachen später die schwierigeren Typen von Feindflugzeugen.

Der Spieler steuert ein großes gelbes Flugzeug, das aufwärts fliegt. Drei der vier gegnerischen Flugzeugtypen sind oben im Bild dargestellt. Im unteren Teil wird der Punktestand, die Anzahl der verbleibenden Leben und der erlittene Schaden (dargestellt durch einen grünen Balken) angezeigt.

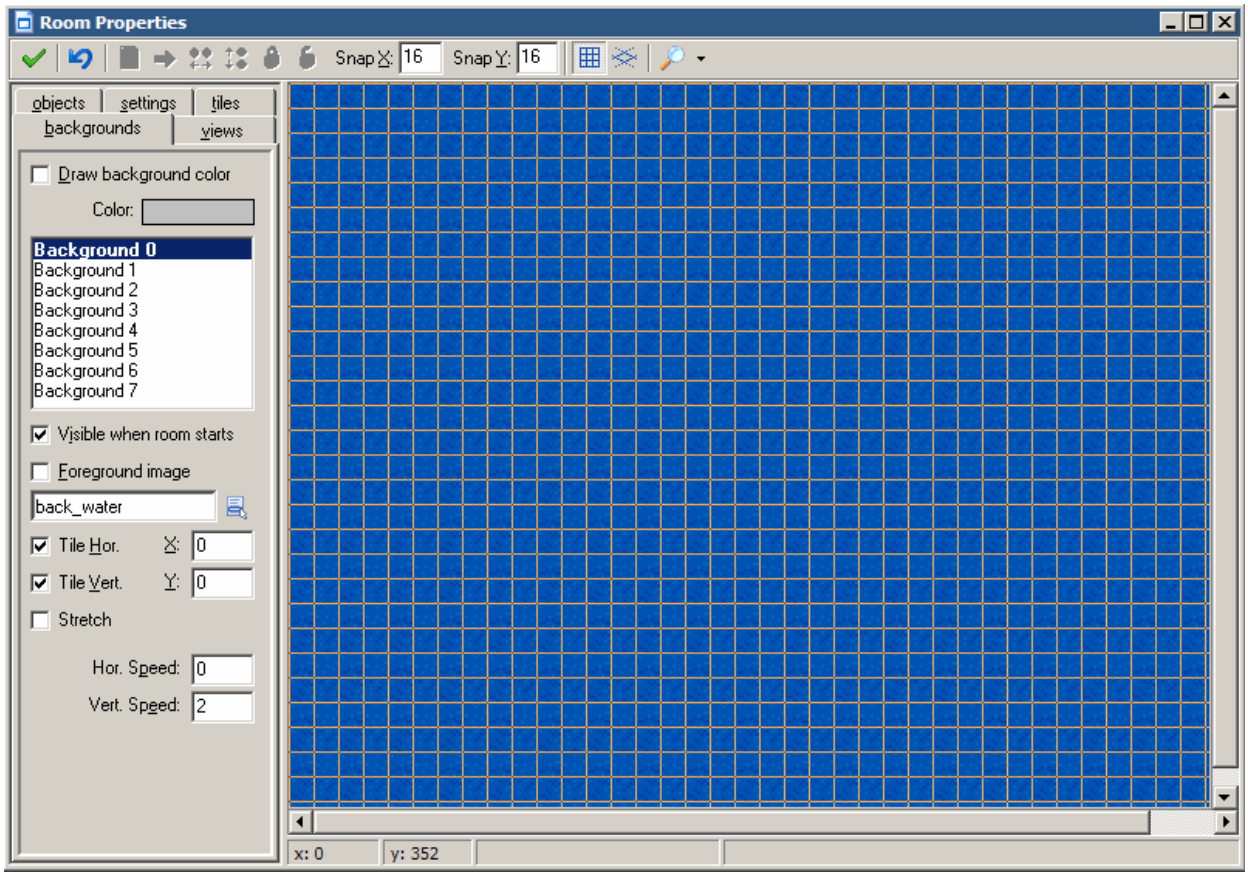
Die Illusion der Bewegung

Ein "scrolling shooter" verdankt seinen Namen der Tatsache, daß die Spielwelt über den Bildschirm rollt (scrolling~Bildschirmrollen) , gewöhnlich von oben nach unten oder von rechts nach links. Das erzeugt die Illusion einer Bewegung. Im Spiel 1945 scrollt die Spielwelt vertikal. Auch wenn das vom Spieler gesteuerte Flugzeug still steht erhälst Du den Eindruck, als ob es über den rollenden Hintergrund flöge. Du kannst die Position des Flugzeuges steuern, indem Du es auf dem Bildschirm umher bewegst.
Dies erzeugt den Eindruck als beschleunige es, wenn es sich vorwärts bewegt oder es würde langsamer, wenn es sich rückwärts bewegt. Es ist entscheidend, daß sich das Flugzeug nicht schneller nach hinten bewegt, als der Bildschirm scrollt. Dies würde den Eindruck erwecken, als flöge das Flugzeug rückwärts, was natürlich unmöglich ist.
So, wie erstellen wir jetzt einen scrollenden Hintergrund im Game Maker? Da gibt es eigentlich zwei Möglichkeiten. Die erste und einfachste ist, einen gekacheltes Hintergrundbild zu verwenden, welches sich nach unten durch den Raum bewegt. Der zweite, etwas kompliziertere, Weg ist einen viel größeren Raum zu erstellen aber nur einen Teil dieses Raumes anzuzeigen, indem man einen sogenannten "view" benutzt. Dieser view bewegt sich langsam aufwärts über den Raum. Wir werden anfangen mit einem bewegten Hintergrund. Später gehen wir kurz auf die zweite Option ein.

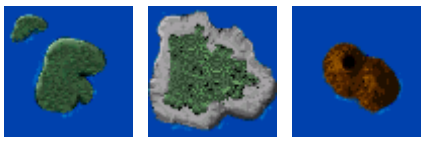
Da unser Spiel über dem Meer stattfindet benötigen wir ein Hintergrundbild, was so aussieht, wie ein Ozean von oben. Wir fügen folgendes kleine Bild als eine Hintergrundkomponente dem Spiel hinzu und benennen es als `back_water`:

Es ergibt eine ansehnliche See, wenn der Hintergrund mit ihm angefüllt wird. Um einen Raum mit einem bewegten Hintergrund zu erstellen, füge dem Spiel einen Raum hinzu, wie sonst auch.

Auf der linken Seite klickst Du auf den Reiter, der mit "**backgrounds**" beschriftet ist. Wir müssen da drei Einstellungen verändern. Zuerst, weil wir den ganzen Raum ausfüllen möchten und deshalb keine Hintergrundfarbe brauchen, deaktiviere das Kästchen "**Draw background color**".
Zweitens klicke in der Mitte auf das Menüicon und wähle das "`back_water`" Hintergrundbild. Voreingetellt ist, daß das Bild über den Raum gekachelt wird, was wir auch wollen. Zuletzt müssen wir den Hintergrund noch bewegen. Setze dafür die "**Vert. speed**" auf 2 (befindet sich unten). Das ganze sollte jetzt wie folgt aussehen:

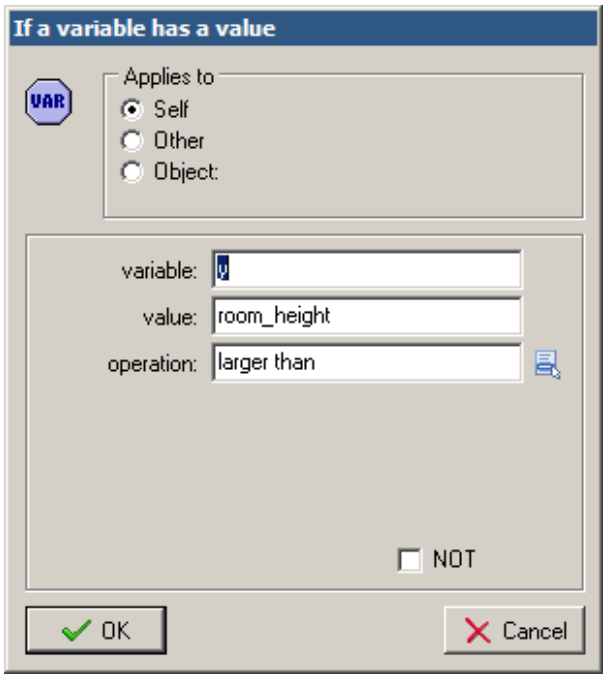


Starte das Spiel (Run), um zu überprüfen, ob wir tatsächlich einen "scrollenden" Hintergrund haben, der den Eindruck von Bewegung vermittelt. Damit das Gefühl der Bewegung noch verstärkt wird, werden wir noch einige Inseln dem Meer hinzufügen. Ein einfacher Weg wäre es, ein größeres Hintergrundbild zu erstellen und eine Insel dort einzufügen. Der Nachteil dieser Herangehensweise ist, daß die Insel in einem regelmäßigen Muster auftaucht, was der Spieler rasch bemerkt. (Du magst so etwas schon einmal gesehen haben in Cartoons/Zeichentrickfilmen, wo ein sich bewegender Hintergrund hinter einer rennenden Figur wiederholt wird.) Deshalb wählen wir einen anderen etwas komplizierteren Ansatz und fügen die Inseln als Objekte hinzu. Zuerst erstellen wir drei sprites mit folgenden Bildern (mit transparentem Hintergrund, wie es voreingestellt ist):



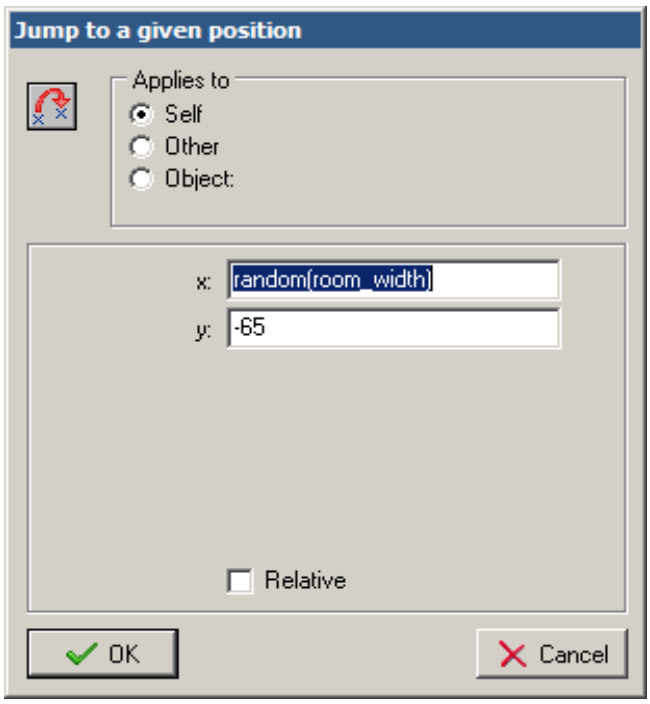
Da wir sie niemals für eine Kollisionserkennung benötigen, deaktivierst Du am Besten das Kästchen "**Precise collision checking**". Für jede der Inseln erstellen wir ein Objekt. Im "creation event" setzen wir die vertikale Geschwindigkeit, die genauso groß ist, wie die Scrollgeschwindigkeit des Hintergrundes.

Dadurch sieht es so aus, als ob die Inseln Bestandteil des Hintergrundes sind. Damit gewährleistet ist, daß alle anderen Objekte sich oberhalb der Inseln befinden, setzen wir die "**depth**" (Tiefenebene) der Insel-Objekte auf 10000. (Der Game Maker muß im "advanced mode" gestartet sein, damit das möglich ist!) Die Instanzen der Objekte werden in der Reihenfolge ihrer "depth" gezeichnet. Die Instanzen mit der größten "depth" werden zuerst gezeichnet. Instanzen mit kleinerer "depth" werden obendrauf gezeichnet. Indem wir den Inseln eine große "depth" geben, werden sie immer zuerst gezeichnet und liegen somit unterhalb der anderen Objekte.
Eine Sache muß noch erledigt werden. Wenn die Inseln an der Unterkante des Raumes verschwinden wollen wir, daß sie erneut an der Oberkante erscheinen. Damit wir das erreichen, prüfen wir im "**step event**" der Insel, ob die Insel an der Unterkante des Bildschirms verschwunden ist und wenn das so ist, laßen wir sie wieder an der Oberkante auftauchen. Beachte, daß die Variable y die vertikale Poisition der Instanz angibt. Die Variable "`room_height`" gibt die Höhe des Raumes an. Deshalb verschwindet die Insel an der Unterkante, sobald y größer als "`room_height`" ist. Demnach können wir die Aktion benutzen, die Variablenwerte prüft, um festzustellen, ob die Insel unterhalb des Raumes liegt:



Wie Du siehst, kannst Du als Wert auch eine andere Variable angeben. Du kannst ganze Ausdrücke hier eingeben.

Um zur Oberkante des Raumes zu gelangen verwenden wir die Aktion "jump to a position". Aber wir wollen zu einer zufälligen Position oberhalb des Raumes springen, nicht zu einer bestimmten. Durch das Auswählen einer zufälligen Position erreichen wir weniger Regelmäßigkeit bei den Stellen, wo die Inseln erscheinen. Dadurch hat der Spieler nicht das Gefühl, daß es die gleiche Insel, die auftaucht. Aber wie erhalten wir einen zufälligen Wert? Dafür gibt es eine Funktion **random()**. Du fragst vielleicht, was eine Funkton ist? Eine Funktion berechnet einen Wert (oder führt eine Aktion aus) basierend auf Werten von bestimmten Argumenten. Die Argumente werden zwischen die Klammer gesetzt, welche dem Funktionsnamen folgen. Überall, wo Du Werte eingeben kannst, laßen sich auch Funktionen und Variablen angeben (und Ausdrücke, die sie enthalten). Es gibt viele Funktionen im Game Maker. Aber im Moment benötigen wir nur die Funktion random. In der "jump to a position"-Aktion verwenden wir **random(room_width)**. Dies erzeugt einen Wert zwischen 0 und der Breite des Raumes, so wie wir es wollen. Demnach sieht die jump-Aktion jetzt so aus:



Wir geben -65 für die y-Position an, um sicherzustellen daß die Insel komplett oberhalb des Raumes startet. Aufgrund ihrer vertikalen Geschwindigkeit wird sie sich wieder in den sichtbaren Bereich bewegen.

Alles was noch gemacht weden muß, ist die Insel auf unterschiedliche Höhen (heights) im Raum zu setzen und schon ist unser scrollender Hintergrund fertig. Obwohl die Inseln in regelmäßigen Abständen wieder auftauchen, wird der Spieler nicht wirklich bemerken, daß es immer die gleichen sind, weil sie an verschiedenen Stellen erscheinen. Du kannst auch eine gewisse Unregelmäßigkeit beim Erscheinen der Inseln erzeugen, indem Du einen (negativen) zufälligen Wert für die y-Koordinate verwendest.

Das Spielerflugzeug

Nun, da der scrollende Hintergrund bereit steht, ist es an der Zeit, daß Flugzeug zu erstellen, welches der Spieler steuern wird. Das ist in der Tat sehr einfach. Zuerst brauchen wir ein sprite für das Flugzeug. Wir werden ein Propellerflugzeug benutzene. Um den Eindruck eines sich drehendes Propellers zu erwecken, verwenden wir ein sprite bestehend aus drei Einzelbildern Bilder, die exakt gleich sind, bis auf die Propeller:

Beim Erschaffen dieses sprites machen wir eine wichtige Sache. Wir setzen **X** und **Y** des Bezugspunktes (**origin**) auf 32. Das bedeutet, daß der Bezugspunkt des sprites in der Mitte des Flugzeuges liegt. Wann immer wir später die Position des Flugzeuges festlegen oder die Position abfragen, beziehen wir uns auf die Mitte des Flugzeuges, nicht auf die linke obere Ecke, die normalerweise als Bezugspunkt genutzt wird. Das ist wichtig um sicherzustellen, daß beispielsweise Kugeln aus der Mitte des Flugzeuges kommen und nicht von der linken Seite.

Als nächstes fügen wir das Objekt `obj_myplane` hinzu. Als sprite verwenden wir das vorhin erstellte Flugzeugsprite. Wir weisen ihm die depth von -100 zu, um zu gewährleisten, daß es oberhalb von Kugeln usw. liegt, die wir später noch erstellen. (Der Umgang mit der depth-Eigenschaft in effektiver Weise ist sehr wichtig für viele Spiele. Darum ist es besser, wenn Du den Umgang damit verstehst.)
Im Augenblick müssen wir nur die Bewegung des Flugzuges beschreiben. Wenn der Spieler nichts macht, wird sich das Flugzeug nicht bewegen. (Erinnere Dich, der Hintergrund bewegt sich, nicht das Flugzeug.) Wenn der Spieler eine der vier Pfeiltasten drückt, sollte sich das Flugzeug in die entsprechende Richtung bewegen. Unser Hauptanliegen ist zu vermeiden, daß das Flugzeug sich aus dem Raum hinaus bewegt. Aus diesem Grund werden wir die Konrolle über die Bewegung des Flugzeuges selber übernehmen, anstatt dem Flugzeug eine Geschwindigkeit zuzuweisen. Das wird wie folgt gemacht.

Laß uns auf die Bewegung für die linke Pfeiltaste blicken. Zuerst müssen wir überprüfen, ob wir nicht zuweit links sind. Deshalb verwenden wir die Aktion um zu prüfen, ob die Variable **x** größer als 40 ist, ähnlich wie beim Überprüfen der y-Koordinate von den Inseln oben. Wenn ja benutzen wir die Aktion "jump to a position" für die relative Bewegung mit -4 für x und 0 für y. Vergiß nicht ein Häkchen bei der Checkbox "**Relative**" zu machen. Wir wollen es ein kleines Stückchen nach links bewegen, relativ zur aktuellen Position. Für die echte Pfeiltaste machen wir eine ähnliche Sache. Wir prüfen, ob **x** kleiner als `room_width`-40 ist; wenn ja, springen wir relativ zu einem x von 4 und einem y von 0. Ähnlich für die vertikale Bewegung aber diesmal bewegung wir uns nur mit einer Geschwindigkeit von -2 und 2. (Erinnere Dich, wir sollten uns niemals schneller rückwärts bewegen, als der Hintergrund scrollt.) An der Unterkante des Bildschirmes sparen wir einen Rand von 120. Dieser wir später verwendet für ein Anzeigefeld mit Spielinformationen.

Unser Flugzeug kann nun fliegen. Setze eines in den Raum und fahre fort. Wenn Du es nicht selber machen willst, findest Du eine Datei namens `1945_1.gmd`, die das bisherige Spiel beinhaltet.

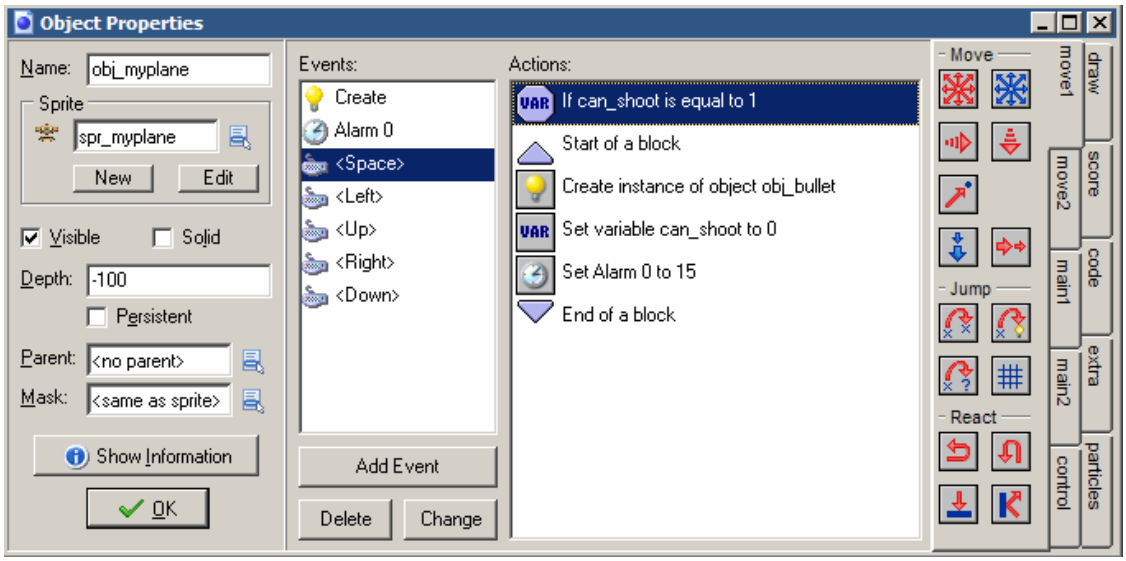
Feinde und Waffen

Was taugt ein "scrolling shooter", wenn Du nicht schießen kannst und keine Gegner vorhanden sind? Wir werden nun unser Spiel mit einigen Gegnern erweitern und einer Kanone, die Du abfeuern kannst. Du findest diese Version des Spiels in der Datei `1945_2.gmd`.

Laß uns mit der Kanone anfangen. Wir brauchen ein sprite für die Kugel. Wir setzen de Bezugspunkt in die MItte, wie wir es schon beim Flugzeug getan haben. (Du kannst das auch erreichen, indem Du mit der Maus an die entsprechende Stelle in der Abbildung rechtsvom Spriteformular klickst.) Um es etwas spektakulärer zu gestalten, verwenden wir eine ziemlich große Kugel. Übertreiben von Dingen ist oftmals wichtig in Spielen.

Wir erstellen ein Objekt mit diesem sprite. Wir geben ihm die voreingestellte depth von 0, so daß es unter dem Flugzeug und oberhalb der Inseln erscheint. Das Objekt hat ein recht einfaches Verhalten. Im "creation event" geben wir ihm eine vertikale Geschwindigkeit von -8, damit es sich aufwärts bewegt. Um zu vermeiden, daß mehr und mehr Kugeln umher fliegen, müssen wir sie zerstören, sobald sie den Raum verlassen. Das kann sehr einfach erreicht werden. Im "step event" prüfen wir, ob die Variable **x** kleiner als -16 ist. Du solltest mittlerweile wissen, wie man das anstellt. Wenn es so ist, zerstören wir die Objekte durch die entsprechende Aktion. (Du kannst auch das "**Outside room event**" verwenden, welches bei den anderen "events" zu finden ist.)

Die Kugel soll abgefeuert werden, wenn der Spieler die Leertaste drückt. Wie in den meisten "Shootern" soll das Flugzeug solange feuern, wie die Taste gedrückt wird. Aber wir wollen nicht zuviele Kugeln zur gleichen Zeit. Das würde das Spiel zu einfach gestalten. Wir erlauben dem Spieler nur zwei Kugeln pro Sekunde abzufeuern, das heißt eine Kugel alle 15 "steps". Um das zu erreichen nutzen wir, wie schon erwähnt, eine Variable `can_shoot`. Im "creation event" des Spielerflugzeuges setzen wir diese Variable auf 1, um anzuzeigen, daß wir tatsächlich eine Kugel abfeuern können. Im "space key event" prüfen wir, ob die Variable `can_shoot` gleich 1 ist. Wenn ja, erstellen wir eine Kugel genau vor dem Flugzeug, relative an Position (0,-16). Zudem setzen wir die Variable `can_shoot` auf 0, um anzuzeigen, daß wir nun nicht mehr schießen können und setzen auch noch alarm0 auf 15. Erinnere Dich, ein Alarm tickt runter auf 0 - ein Tick pro "step". Wenn der Alarm auf 0 runter getickt ist, wird das "alarm event" ausgelöst. In diesem "event" setzen wir die Variable `can_shoot` wieder auf 1, um anzuzeigen, daß wir wieder schußbereit sind. Somit sieht das "event" für die Leertaste ungefähr so aus:



Du kannst die Schußfrequenz ändern, indem Du den Wert der Alarmuhr (alarm clock) änderst. (In einigen Spielen kannst Du schneller schießen, wenn Du ständig die Leertaste drückst. Dies kann durch einen ähnlichen Satz von Aktionen im "space key pressed event" (Leertaste gedrückt Ereignis) erreicht werden - nur diesmal ohne die Variablenüberprüfung.)

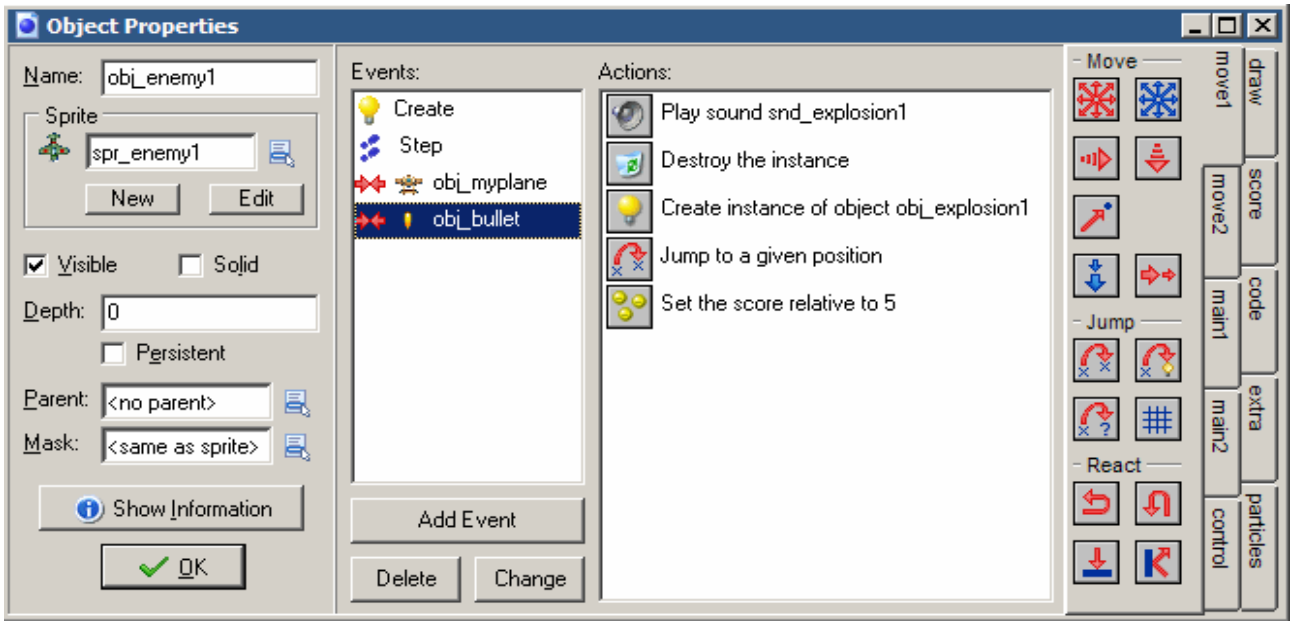
Laß uns nun den ersten Gegner erstellen. Dieser ist ein kleines Flugzeug, was einfach abwärts fliegt. Es schießt nicht aber wenn es mit dem Spielerflugzeug zusammenstößt, ist das Spiel verloren. Wieder erstellen wir ein sprite für das Feindflugzeug und ein Objekt. Im creation event setzen wir die vertikale Geschwindigkeit auf 4, damit es abwärts fliegt. Sobald das Flugzeug die Unterkante des Raumes erreicht, lassen wir es an der Oberkante des Raumes an einem zufälligen Ort

wiedererscheinen, genau so, wie wir es bei den Inseln gemacht haben. Mittlerweile solltest Du wissen wie das geht.

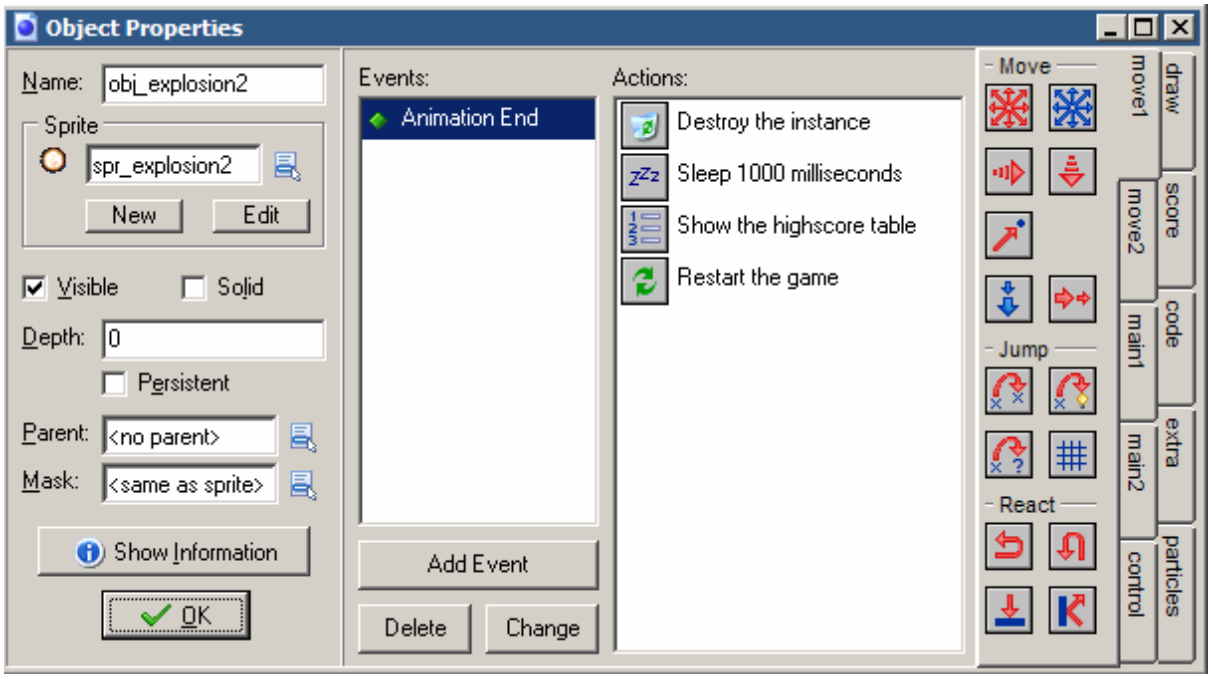
Wir müssen zwei wichtige "collision events" (Kollisionsereignis) für das Feindflugzeug festlegen: Das Kollisionsereignis mit der Kugel, die das Feindflugzeug zerstören soll und das Kollisionsereignis mit dem Spielerflugzeug, welches das Spielerflugzeug zerstören soll und das Spiel beendet.

Wir beginnen mit dem Kollisionsevent für die Kugel. Ein Menge von Aktionen wird hier benötigt. Aber zuerst brauchen wir ein Geräusch für die kleine Explosion und ein sprite, um die Explosion anzuzeigen. Um ein Geräusch zu erstellen, klicke auf "**Add sound**" (Füge Geräusch hinzu) und lade ein nettes Explosionsgeräusch.
Weil wir möglicherweise das Geräusch mehrfach zur gleichen Zeit benötigen, setzen wir die "buffer" (Pufferspeicher) auf 4. Das bedeutet, daß vier Kopien des Geräusches simultan wiedergegeben werden können. Für die Explosion brauchen wir ein kleines sprite. Wie für das Feindflugzeug setzen wir den Bezugspunkt (origin) auf (16,16). Zusätzlich erstellen wir ein Explosion-Objekt und weisen ihm das Explosions-sprite zu. Es macht nichts weiter, als sich nach der Explosionsanimation selber zu zerstören. Dafür gibt es ein event namens "**Animation end**", zu finden bei den anderen events.

Sobald das Geräusch und das Explosions-Objekt fertig sind, können wir das Kollisionsereignis des Feindflugzeuges mit der Kugel einfügen. Dafür brauchen wir folgende Aktionen. Zuallererst spielen wir den Explosions-sound ab. Dann vernichten wir die Kugel. Dafür benutze die Aktion "destroy an instance" aber, gib oben in dem Formular an, es für die andere Instanz gilt (**other instance**), was in diesem Fall die Kugel ist. Als nächstes erstellen wir das Explosions-Objekt relativ an Position (0,0), was der Stelle des Feindflugzeuges entspricht. Wir zerstören das Feindflugzeug nicht! Stattdessen bewegen wir es an eine zufällige Stelle oberhalb des Raumes, so daß es aussieht, als würde ein neues Flugzeug ankommen. Schließlich setzen wir den score (Punktestand) relativ auf 5. Wir brauchen das "relative" hier, weil wir dem Punktestand fünf Punkte hinzufügen möchten; wir wollen den Punktestand nicht auf 5 setzen. Das Ereignis sollte jetzt etwa so aussehen:



Wir müssen noch die Kollision mit dem Spielerflugzeug behandeln. Wieder brauchen wir ein Explosions-sprite (ein bischen größer diesmal) und ein Explosions-Geräusch (ein wenig lauter diesmal).
Wir fügen den Sound dem Spiel hinzu (diesmal brauchen wir keine buffer - das Spielerflugzeug kann nur einmal explodieren) und wir machen ein sprite für die Explosion. Wieder erstellen wir ein Explosions-Objekt dafür aber diesmal ist das Objekt ein bischen mehr Arbeit, als die anderen Explosions-Objekte, weil es auch noch das Ende des Spiels verwaltet. In seinem "**animation end event**" stellen wir ein paar Sachen an. Zuerst zerstören die Instanz, um sie unsichtbar zu machen. Dann machen wir eine kleine Pause, um sicherzugehen, das der Explosions-Sound beendet ist. Die nächste Aktion zeigt die High-score Tabelle, so daß der Spieler seinen/ihren Namen eingeben kann, wenn ein Highscore erreicht wurde. Das geht alles automatisch im Game Maker. Du kannst das Erscheinungsbild der Highscore Tabelle selber festlegen. Du kannst ein hübsches Hintergrundbild haben, die Schriftart auswählen, usw.. Du solltest damit ein wenig experimentieren. Schließlich starten wir das Spiel erneut. Damit sieht das event wie folgt aus:



Im Kollisionsereignis vom Feindflugzeug mit dem Spielerflugzeug, ändern wir das Spielerflugzeug - das ist die "other instance" - in eine Explosion. Zudem spielen wir das Explosionsgeräusch ab und zerstören das Feindflugzeug.

Was noch bleibt, ist ein Feindflugzeug im Raum zu platzieren aber wir werden das etwas anders machen. Ein gutes Spiel wird mit der Zeit herausfordender. Somit beginnen wir mit einem Feindflugzeug und im Lauf der Zeit kommen weitere hinzu. Dafür erstellen wir ein weiteres Objekt, das wir "`controller_enemy`" nennen werden. Es wird die Erzeugung von Feindflugzeugen steuern. Wir machen dieses Objekt während des Spieles unsichtbar, indem wir die Kontrollbox, die mit "**visible**" bezeichnet ist, deaktivieren. Ein sprite benötigen wir nicht dafür. Im "creation event" erstellen wir ein Feindflugzeug an einer zufälligen Stelle knapp oberhalb des Raumes. Das wird der erste Gegner sein. Zusätzlich setzen wir eine "alarm clock" auf 200. Im "event" für diese "alarm clock" erstellen wir ein weiteres Feindflugzeug und setzen die "alarm clock" erneut aber diesmal auf 500. Das ist alles. Der Effekt besteht darin, daß zu Beginn des Spieles ein Feindflugzeug vorhanden ist. Nach 200 "steps", das sind ungefähr 7 Sekunden, erscheint ein zweites Feindflugzeug. Nach ungefähr 15 Sekunden erscheint ein drittes Flugzeug und so weiter. (Die Tatsache, daß das zweite Flugzeug eher auftaucht, als die anderen, liegt daran, daß das Spiel zu langweilig ist mit nur einem

Flugzeug.) Setze eine Instanz dieses Objektes und wir sind fertig.

Das beendet die zweite Version von 1945. Wir haben ein spielbares Spiel mit einem Gegner.

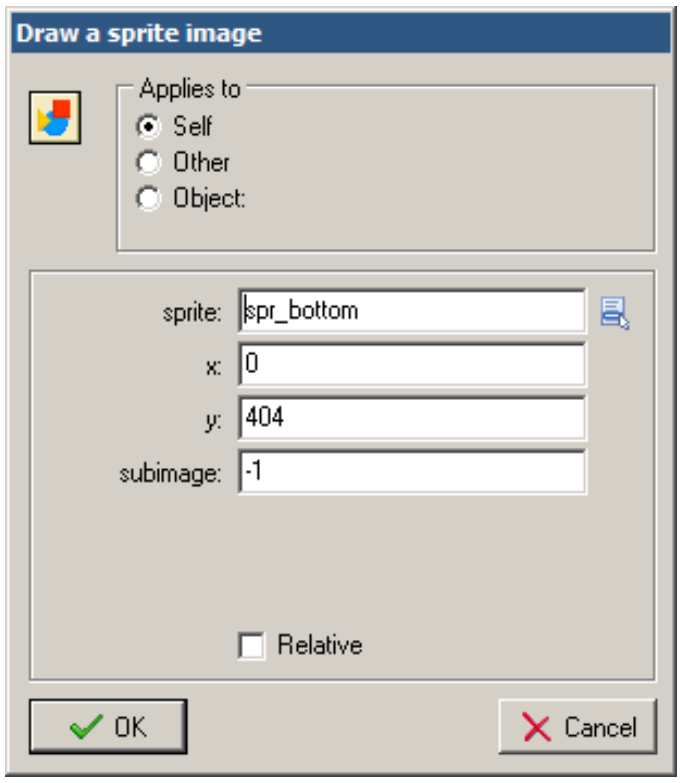
Punkte, Leben und Schaden

Es ist ein bischen unbefriedigend, daß das Spiel endet wann immer man getroffen wird. Um das Spiel etwas interessanter zu gestalten, lassen wir die Gegner Schaden verursachen. Nur wenn das Flugzeug zuviel Schaden nimmt, wird es zerstört. Zudem führen führen wir mehrere Leben ein und erstellen eine hübsche Anzeigetafel, welche diese Informationen, zusammen mit dem Punktestand, darstellt. Glücklicherweise wird all dies sehr einfach sein, da der *Game Maker* eingebaute Mechanismen besitzt, die sich mit Punktestand, Leben und Gesundheit (was das Gegenteil von Schaden ist) befassen. Das neue Spiel ist in der Datei `1945_3.gmd` enthalten.

Damit wir all das erstellen können machen wir ein neues Objekt, das wir **`controller_life`** nennen. Es benötigt selbst kein sprite, weil wir die Zeichnung selber bestimmen mittels des "drawing event". Wie Du weißt, wird normalerweise in jedem "step" das gesetzte sprite einer Instanz an der richtigen Position im Raum gezeichnet. Aber wenn Du Aktionen ins "drawing event" setzt, ist das nicht mehr der Fall. Jetzt bestimmen diese Aktionen, was gezeichnet wird. Es gibt eine ganze Sammlung von Aktionen nur für das Zeichnen. Die meisten können unter der Registerkarte "**draw**" gefunden werden . Aber Du kannst auch andere Aktionen hier verwenden. Die "drawing actions" machen nur Sinn im "drawing event". An anderer Stelle werden sie schlicht ignoriert.

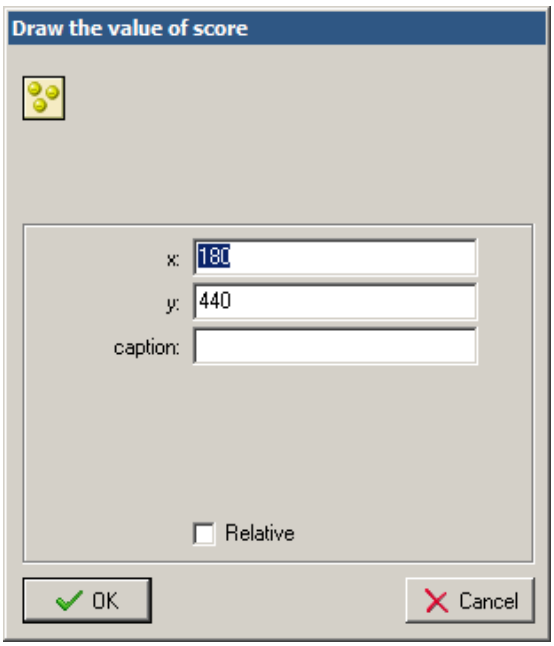
Als Anfang erstellen wir ein großes sprite, das als Informations-Anzeigefeld fungiert. Es sieht so aus:

Es wird den Punktestand darstellen, den Schaden (im schwarzen Bereich) und die Anzahl der verbleibenden Flugzeuge, das ist die Anzahl von Leben. Im "drawing event" von `controller_life` zeichnen wir dieses Informationsfeld-sprite an der korrekten Stelle, indem wir die Aktion "draw a sprite" verwenden. Wir fügen die Parameter wie folgt ein:

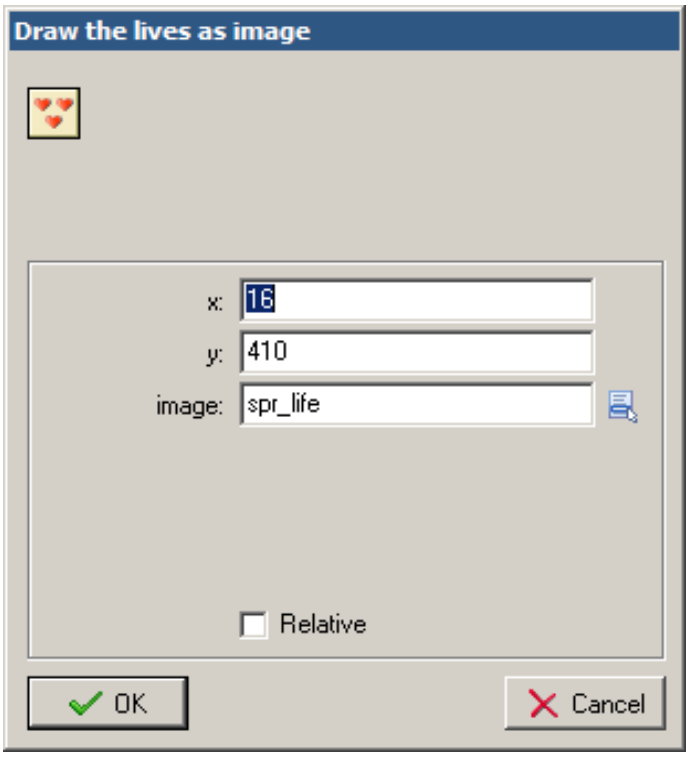


Dies wird das richtige sprite an die Unterkante des Bildschirmes setzen. (verwenden von -1 für das "subimage" (~Einzelbild einer Bildfolge) bedeutet, daß das aktuelle Einzelbild gezeichnet wird. Da dieses sprite nur aus einem Bild besteht, kümmern wir uns nicht wirklich darum, wenn aber ein sprite aus mehreren Einzelbildern besteht, kannst Du hier angeben, welches Du sehen willst.) Damit das Anzeigefeld oberhalb von allem liegt, geben wir dem `controller_life` Objekt eine "depth" von –10000.

Im "creation event" setzt das `controller_life` Objekt den Punktestand auf 0, die Anzahl an Leben auf 3 und die Gesundheit (health) auf 100. Es gibt Aktionen dafür unter der Registerkarte "**score**". Um den Punktzestand darzustellen verwenden wir die entsprechende Aktion auis der "**score**"-Registerkarte. (Zuerst legen wir eine weiße etwas größere Schriftart fest.) Wir geben die Parameter wie folgt an (Keine "caption" (Titelleiste), weil diese schon auf dem Hintergrund ist):

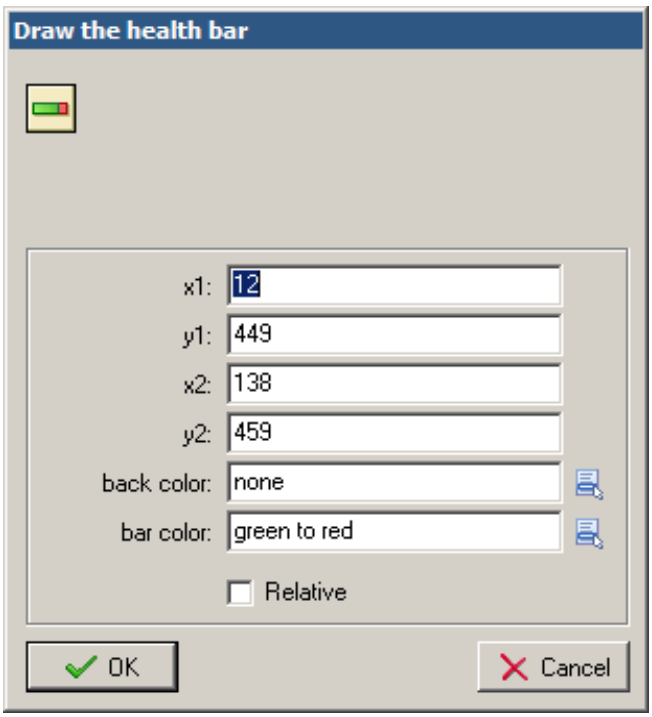


Um die Leben darzustellen, verwenden wir einen anderen Mechanismus. Anstatt nur die Zahl darzustellen, werden wir eine Anzahl von Miniaturbildern des Flugzeuges zeichnen. Dafür verwenden wir ein kleines sprite verwenden, welches das Flugzeug abbildet. Es gibt eine Aktion dafür in der "**score**"-Registerkarte (draw the lives as images).

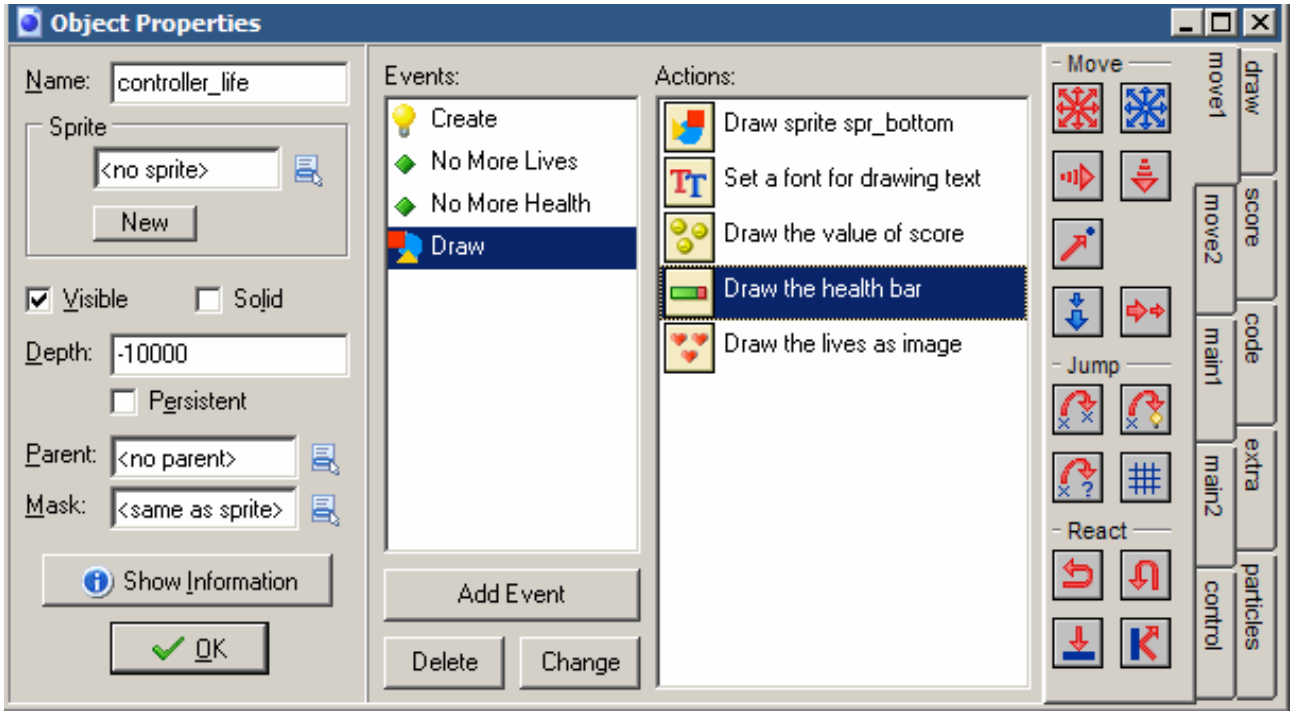


Da wir nun selber für die Darstellung von Punktestand und Lebenanzahl sorgen, möchten wir, daß sie nicht mehr in der Titelleiste dargestellt werden. Es gibt eine Aktion in der "score"-Registerkarte, die angibt, was in der Titelleiste angezeigt werden soll. Setze sie in das "creation event" des Objektes und gib an, daß nichts angezeigt werden soll.

Zum Zeichnen der "health" (Gesundheit) steht auch eine spezielle Aktion zur Verfügung. Die "health" wird in Form einer "health bar" (eine Art Balken) dargestellt. Du kannst die Position, Größe und das zu verwendende Farbschema angeben. Wir geben folgende Paramer an:

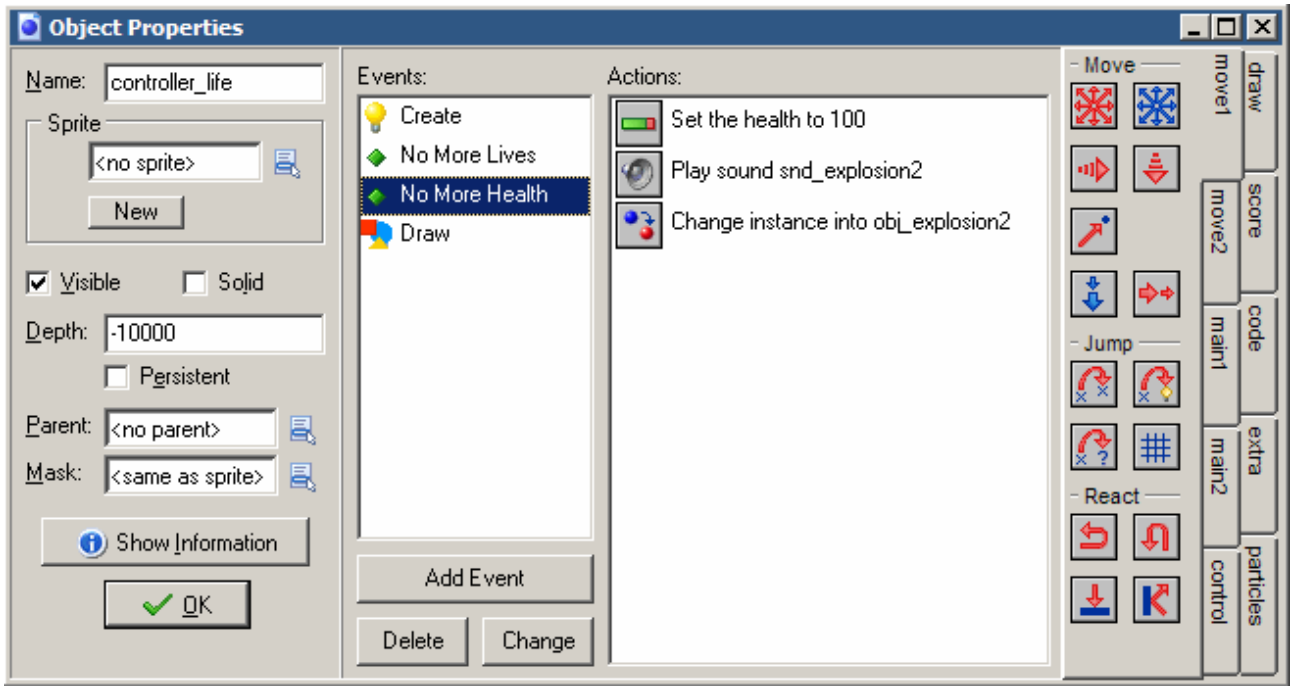


Das ganze "drawing event" sieht dann so aus:



Aber wir müssen immer noch "health" (Gesundheit) und "lives" (Leben) überprüfen können. Zuersteinmal nehmen wir ein paar Änderung im Kollisionsereignis für Feindflugzeug und Spielerflugzeug vor. Es soll nicht mehr das Spielerflugzeug zerstören, sondern nur sich selber

(dabei verwandelt es sich in eine Explosion) und vermindert die "health", das heißt, sie relativ auf -30 zu setzen (somit kann es 3 Treffern widerstehen).
Das `controller_life` Objekt überprüft, ob die "health" kleiner als 0 wird. Dafür gibt es ein "event" bei den anderen "events". Innerhalb dieses "events" jagen wir das Spielerflugzeug in die Luft, indem wir es in eine große Explosion verwandeln. Wir setzen auch die "health" zurück und laßen den richtigen Sound erklingen. Somit sieht das "event" dann so aus:



Das große Explosions-Objekt zerstört sich selber in seinem "**Animation end event**". Pausiert für einen Moment, damit der Sound zu Ende abgespielt werden kann, erstellt dann ein neues Spielerflugzeug an der momentanen Position (das ist relativ an Position (0,0)) und vermindert die Anzahl an "lives" relativ um -1.

Schließlich müssen wir noch prüfen, ob wir alle "lives" aufgebraucht haben. Glücklicherweise gibt es dafür wieder ein "event". Für das controller_life Objekt zeigen wir in diesem "event" die Highscore-Tabelle und starten das Spiel erneut. Dies schließt die dritte Version des Spieles ab. Du kannst es jetzt spielen und es sieht schon recht hübsch aus. Aber bald schon wird es langweilig. Somit müssen wir es noch etwas erweitern und einige Variationen hinzufügen.

Weitere Gegner

In diesem Abschnitt werden wir drei neue Typen von Feindflugzeugen unserem Spiel hinzufügen. Eines wird Kugeln senkrecht nach unten abfeuern. Das zweite wird Kugeln in Richtung des Spielerflugzeuges abfeuern. Der dritte Typ wir keine Kugeln abfeuern aber von der Unterkante des Raumes her auftauchen. Wir werden sie späteren Stufen des Spiels auftauchen lassen. Die neue Version findest Du in der Datei `1945_4.gmd`.

Um den zweiten Typ von Feindflugzeug zu erstellen, machen wir zuerst ein neues sprite dafür, ähnlich dem des ersten Feindflugzeuges aber mit einer anderen Farbgebung. Zweitens benötigen wir ein neues Objekt für es. Weil das Objekt fast das gleiche Verhalten hat, wie das erste Feindflugzeug, machen wir davon eine Kopie (Rechtsklick auf das Objekt und "**duplicate**" auswählen).

Klicke doppelt auf das neue duplizierte Objekt, um es zu ändern.

Gib ihm einen neuen Namen und weise das richtige sprite zu. Da es ein etwas spezielleres Flugzeug ist, sollte der Spieler mehr Punkte dafür bekommen, wenner es abschießt. Deshalb ändern wir im Kollisions event mit der Kugel den "score" auf 10.

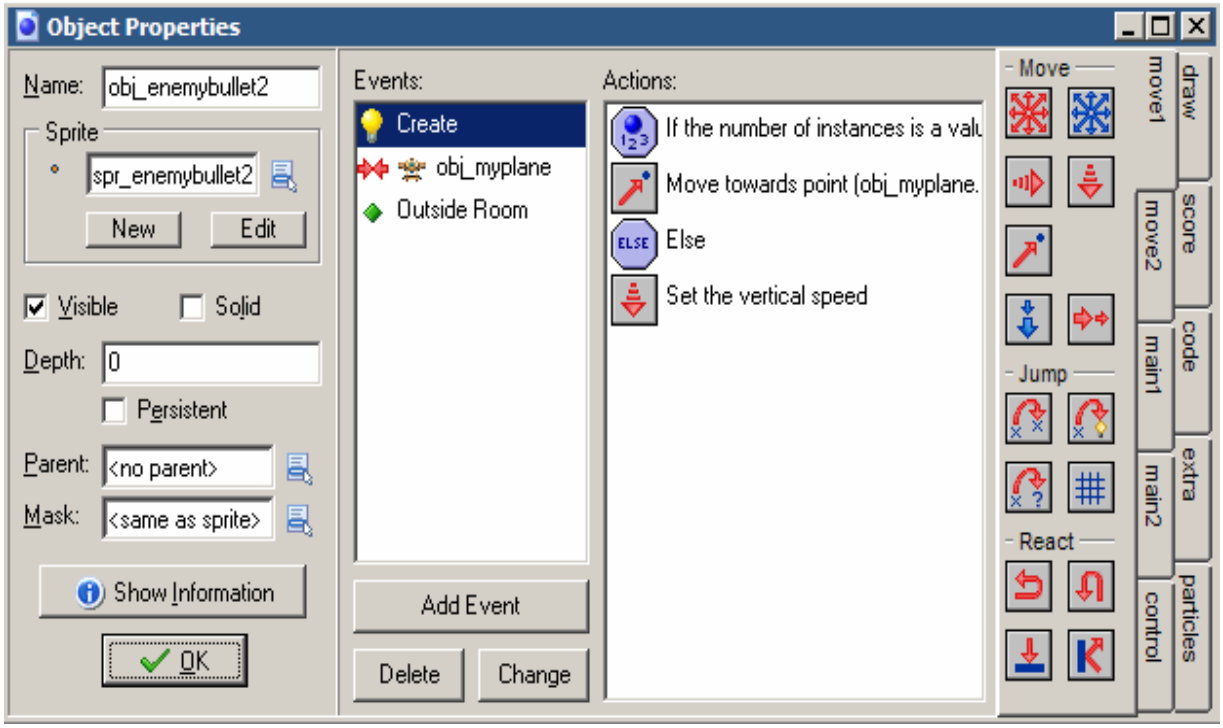
Damit das Feindflugzeug schießen kann, brauchen wir ein Projektil-sprite und ein Projektil-Objekt. Dieses Objekt bekommt in seinem "creation event" eine vertikale abwärts gerichtete Geschwindigkeit zugewiesen. In seinem "step event" kümmern wir uns wieder darum, daß das Objekt zerstört wird, sobald es den Raum an der Unterkante verläßt. Im Kollisions event dieses Projektils mit dem Spielerflugzeug setzen wir die "health" relativ auf -5, zerstören das Projektil und spielen einen Sound ab.

Nun müssen wir noch einrichten, daß das Feindflugzeug von Zeit zu Zeit Projektile abfeuert. Wir erledigen das im "step event" des Flugzeuges. Wir verwenden die Aktion "throw a dice" und als Parameter nehmen wir 30.
Das bedeutet, daß im Schnitt einmal alle 30 "steps" die nächste Aktion ausgeführt wird. In dieser nächsten Aktion erstellen wir das feindliche Projektil.

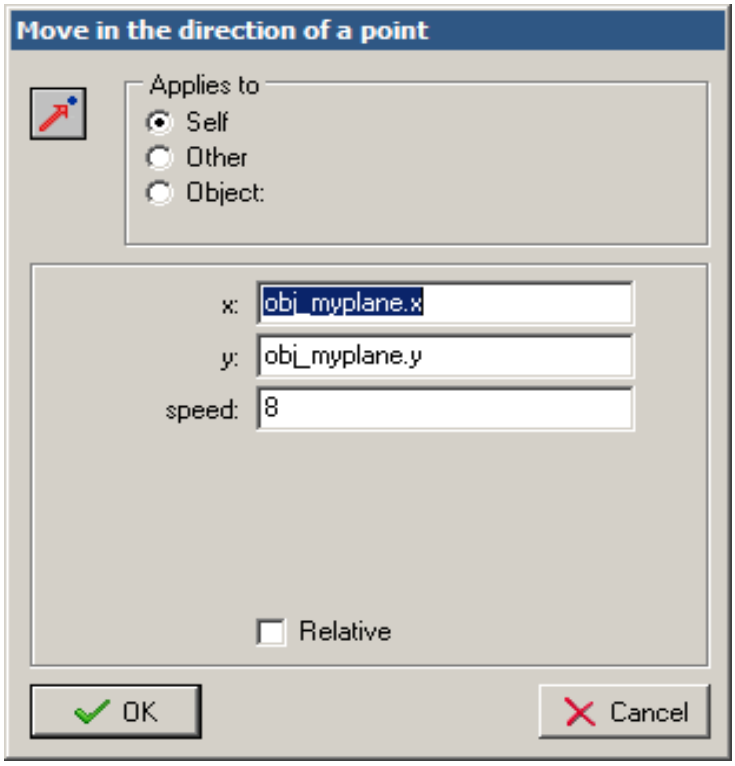
Schließlich müssen wir noch sicherstellen, daß ab einer bestimmten Stufe die zweiten Feindflugzeuge beginnen aufzutauchen. Dafür verwenden wir das `controller_enemy` Objekt nochmal. Im "creation event" setzen wir "alarm1" auf einen Wert von 1000. In diesem "alarm event" erzeugen wir das zweite Feindflugzeug und setzen "alarm1" wieder auf 500, damit wir ein wenig später ein weiteres Flugzeug erzeugen. Somit wird das erste Flugzeug diesen Typs nach ungefähr 30 Sekunden und ein weiteres alle 15 Sekunden erzeugt.

Für unseren nächsten Typ Feindflugzeug brauchen wir wieder ein neues sprite und wir benötigen ein neues sprite für die Kugel. Wir machen eine Kopie vom Objekt des zweiten Feindflugzeuges und , wie oben, im Kollisions event mit der normalen Kugel ändern wir die Punkte auf einen Zuwachs von 20. Zudem erstellen wir ein zweites Feindprojektil-Objekt. Das Feindflugzeug erstellt diese neue Art von Kugel in seinem "step event".
Wieder verwenden wir die "throw a dice"-Aktion aber diesmal erzeugen wir nur alle 80 "steps" eine Kugel, weil dieser neuen Art von Kugel viel schwerer auszuweichen ist.

Die neue Projektilart funktioniert wie folgt. Im "creation event" verwenden wir die Aktion "move towards a position". Aber welche Position sollen wir benutzen? Nun ja, wir wollen in Richtung des Spielerflugzeuges schießen. Somit brauchen wir die x und y Position dieser Objektinstanz. Das kann im Game Maker sehr leicht erreicht werden. Um an den Variablenwert in einer anderen Instanz zu gelangen, stellen wir dem Variablennamen den Namen des Objektes voran. Somit verwenden wir `obj_myplane.x`, um den Wert der x Koordinate des Flugzeuges anzugeben. Wenn mehrere Instanzen eines Objektes vorhanden sind, bekommen wir den Wert in der ersten Instanz. Wenn keine Instanz dieses Objektes vorhanden ist, erhalten wir eine Fehlermeldung. Das kann ein Problem für unser Spiel sein, weil wenn das Flugzeug zerstört ist, ist kurzzeitig kein Flugzeug vorhanden. Deshalb prüfen wir besser vorher, ob das Spielerflugzeug vorhanden ist. Es gibt eine Aktion, die die Instanzen eines bestimmten Objektes zählen kann. Wir verwenden sie, um zu prüfen, ob das Spielerflugzeug verfügbar ist, wenn ja, dirigieren wir die Kugel in Richtung des Flugzeuges. Andernfalls fliegt die Kugel nach unten. Somit sieht das "creation event" wie folgt aus:



Die Aktion "move towards a point" hat nun folgende Parameter:



Eine weitere Änderung wird noch gebraucht. Da die Kugel in irgendeine Richtung fliegen kann, ist es etwas schwieriger im "step event" zu prüfen, ob sie sich außerhalb des Raumes befindet. Aber es gibt ein besonderes "event" dafür: Das "**Outside event**". In dieses "event" setzen wir einfach eine Aktion zum Zerstören des Objekts hinein.

Schließlich müssen wir uns noch mit dem Erzeugen des neuen Typs von Feindflugzeugen beschäftigen. Wie vorher schon angedeutet verwenden wir das `controller_enemy` Objekt dafür. Im "creation event" setzen wir "alarm2" auf 2000. In diesem "alarm event" erstellen wir den neuen Feindflugzeugtyp und setzen "alarm2" wieder auf 1000, um ein weiteres später zu erzeugen.

Damit ist das hinzufügen unseres dritten Flugzeugtyps abgeschlossene. Was noch bleibt, ist hinzuzufügen der von unten anfliegenden Flugzeuge. Das wird in exakt der gleichen Weise gemacht, wie das erste Feindflugzeug, nur daß das Flugzeug unterhalb des Raumes started und nach oben fliegt anstatt nach unten. Das `controller_enemy` Objekt erzeugt sie wieder, diesmal unter Verwendung von "alarm3". Du solltest inzwischen verstanden haben, wie man das macht, aber Du kannst auch im Spiel nachschauen.

Dies beendet unsere vierte Version von *1945*. Es ist nun ein spielbares Spiel geworden, daß mit der Zeit schwieriger wird. Es macht inzwischen ein bischen Spaß es zu spielen und zu versuchen, einen hohen Punktestand zu erreichen.

Fertigstellen des Spieles

Wir haben jetzt einen "scrolling shooter" mit verschiedenen gegnerischen Flugzeugen erstellt. Es ist ein Spielbares Spiel geworden, das mit der Zeit schwieriger wird. Es macht mittlerweile Spaß es zu spielen und zu versuchen, einen möglichst hohen Punktestand zu erreichen. aber um es in ein richtiges Spiel zu verwandeln, müssen noch ein paar letzte Feinheiten erarbeitet werden. Wir benötigen noch Hintergrundmusik, einen Ladebildschirm, ein hübscheres Icon, usw.. Die finale Version des Spiels findest Du in der Datei `1945.gmd`. Ich habe auch eine kleine Verzögerung eingebaut, die abgeschossene Flugzeuge etwas später erscheinen läßt, damit es für den Spieler interessanter ist, die Flugzeuge abzuschießen, statt ihnen einfach nur auszuweichen. Schließlich gibt es noch einen besonders Schuß-Bonus, wenn der Spieler 400 Punkte errreicht und einen weiteren, wenn 1000 Punkte erreicht werden.

Du kannst dieses Spiel als Grundlage nehmen, um es weiter auszubauenr. Hier sind ein paar Ideen, was Du noch einbauen könntest. Es ist üblich in "scrolling shooter" Spielen sog. "power- ups" zu finden, die dem Spieler zusätzliche Feuerkraft verleihen, erlittenen Schaden beheben, zusätzliche Punkte gutschreiben, ein weiteres Flugzeug bescheren, usw..
Solche "power-ups" können auftauchen, wenn der Spieler ein Flugzeug abschießt oder sie begegnen ihm einfach während des Fliegens. Zudem kannst Du noch weitere Feindflugzeuge erstellen, beispielsweise Flugzeuge, die seitlich anfliegen oder welche die stärkere Raketen abfeuern. Ich überlasse es Dir, an diesen Dingen zu arbeiten, um Deinen eigenen "scrolling shooter" zu gestalten. Im Ordner 1945_sprites findest Du eine große Sammlung von sprites, entworfen von Ari Feldman, welche sich besonders für dieses Spiel eignen. Du darfst sie verwenden, um Dein Spiel zu erweitern.

Der Gebrauch von "time lines" (Zeitleisten)

In diesem und im nächsten Abschnitt diskutieren wir zwei weitere Eigenschaften des Game Makers, die sehr hilfreich sind, wenn man kompliziertere "scrolling shooter" erstellen möchte: "time lines" und "views".

Im bisher erstellten Spiel wurde das Erscheinen von Gegnern durch das `controller_enemy` Objekt gesteuert, welches von Zeit zu Zeit Feindflugzeuge dem Spiel hinzufügte. Auch wurden Feindflugzeuge nie richtig vernichtet.
Sie tauchten einfach wieder auf. Das Ergebnis war ein Spiel, daß immer schwieriger wurde, weil die Anzahl von Feindflugzeugen immer mehr wuchs. Obwohl dies ein akzeptables "game play" erzeugt, wirft es einige Probleme auf. Zuersteinmal tauchen die Flugzeuge an zufälligen Stellen auf, die vom Spielentwickler nicht kontrolliert werden können. Als Konsequenz tauchen Flugzeuge übereinander oder sehr nahe am Rand auf, was nicht sehr ansprechend ist. Auch gibt es wenig Überraschungen im Spiel. Gegner kommen auf zufälligen Wegen auf Dich zu und alles was der Spieler machen kann ist ausweichen und die ganze Zeit schießen.

Um ein interessanteres "game-play" zu erwirken, sollte der Spieleentwickler mehr Einfluß darauf haben, wann welches Flugzeug erscheint. Das erlaubt Dir Feindflugzeuge zu erstellen, die in Formationen anfliegen, um bestimmte Herausforderungen zu stellen (wie eine große Gruppe von einfachen Gegner, die man tunlichst meiden sollte oder ein paar gefährliche Gegner, die sehr gut schießen können) und Variationen. Zusätzlich ermöglicht es dem Spieler zu lernen, wie man spielt, weil er sich eventuell an nützliche Sachen aus vorherigen Spielen erinnert. Alle guten Spiele kontrollieren die Gegner, aus eben genannten Gründen, recht genau. Verwende es, es bringt einen interessanteren Spielablauf.

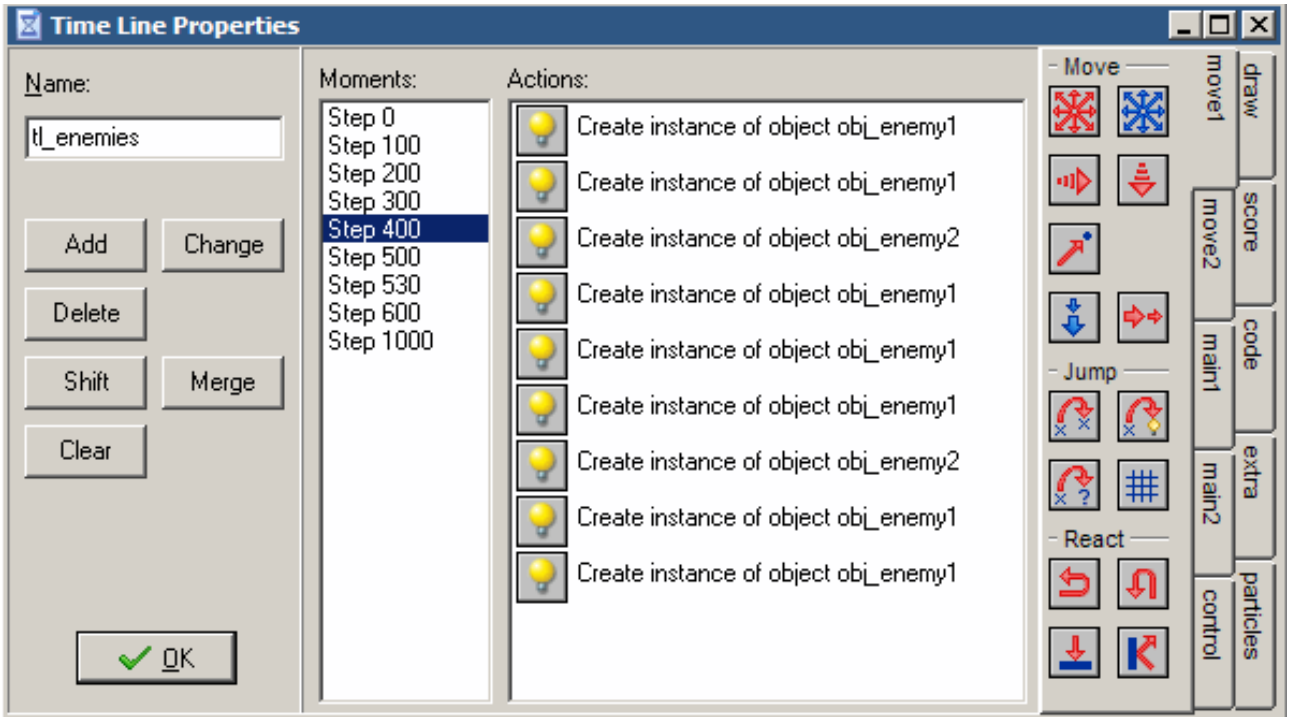
Wir wollen also bessere Kontrolle über das Erscheinen bestimmter Gegner. Dafür verwenden wir "time lines", die ab Version 5 vom Game Maker eingeführt wurden. Aber zuerst müssen wir noch ein paar Äderungen am Spiel vornehmen. Alle Feindflugzeuge tauchen nicht mehr wiederholt auf, wenn sie abgeschoßen werden oder den Bildschirm verlassen. Wir zerstören sie einfach. Wenn nun ein Flugzeug vernichtet wurde, ist es für immer weg. Die "time line" wird das Erzeugen von Gegnern steuern. (Du findest das Spiel in der Datei `1945_5.gmd`. Es ist kein komlettes Spiel. Nur der Anfang wird gezeigt. Du mußt es stark erweitern, um es in ein interessantes Spiel zu verwandeln.)

Eine "time line" Komponente funktioniert global wie folgt. Innerhalb spezifizierst eine Anzahl von Momenten/Zeitpunkten (gemessen in "steps" des Spieles) und für jeden dieser Zeitpunkte kannst Du Aktionen angeben, die dann ausgeführt werden müssen. Weil wir die "time line" für das Erzeugen der Feindflugzeuge verwenden, benutzen wir nur Aktion zum Erstellen von Feindflugzeugen aber normalerweise kannst Du jede Aktion hier einsetzen, die Du magst.

Laß uns jetzt unsere "time line" erstellen. Wähle "Add time line" aus dem "Add menu". Ein Formular erscheint, das so teilweise ähnlich ausschaut, wie das Objekt-Formular. Links kannst Du den Namen der "time line" angeben und es gibt Schaltflächen zum Hinzufügen (add), Löschen (delete) und ändern der Zeitpunkte.
In der Mitte siehst Du eine Liste der definierten Zeitpunkte. Nebenan ist eine Liste der Aktionen des momentan ausgewählten Zeitpunktes und rechts davon die Registerkarten mit den auswählbaren Aktionen.

Zum Zeitpunkt 0 wollen wir ein Feindflugzeug erzeugen. Klicke also auf die Schaltfläche "add". Als Moment/Zeitpunkt gib den Wert 0 ein. Der Zeitpunkt wird der Liste zugefügt. Jetzt kannst Du Aktionen in die Aktionsliste rüberziehen. Wir benötigen nur eine Aktion, um ein Flugzeug an der entsprechenden Stelle zu erstellen. Es braucht ungefähr 100 "steps" für ein Feindflugzeug den Raum zu durchqueren, daher setzen wir einen zweiten Zeitpunkt an "step" 100. Hier erstellen wir zwei Feindflugzeuge nebeneinander. Auf diese Weise fahren wir fort. Wir fügen Zeitpunkte und "creation actions" für Feindflugzeuge hinzu. Durch das Erzeugen von Flugzeugen an unterschiedlichen Stellen oberhalb des Raumes, gelingt es uns nette Formationen zu erstellen.

Nachdem wir nun eine Anzahl von Zeitpunkten erstellt haben sieht das Formular wie folgt aus:



Zum markierten Zeitpunkt 400 erzeugen wir eine Reihe von 9 Feindflugzeugen, wovon zwei schießen können.

Du solltest das Prinzip jetzt verstanden haben. Füge einfach weitere Zeitpunkte mit mehr und mehr gefährlichen Flugzeugen in interassanten Formationen ein und setze die Zeitpunkte eventuell etwas dichter. Es braucht etwas Arbeit eine vollständige Sequenz von Zeitpunkten zu erstellen, die einen netten Level des Spiels darstellen. In einem kompletten Spiel wirst Du verschieden "time lines" für unterschiedliche Level verwenden.
Am Ende der Sequenz mußt Du einen Zeitpunkt einfügen, der den Level oder das Spiel beendet. Hier solltest Du wahrscheinlich eine Nachricht einblenden, das der Level beendet ist oder besser noch eine hübsche End-Sequenz bereithalten, wie beispielsweise einen Flugzeugträger auf dem das Flugzeug landet.

Wir sind noch nicht fertig. Eine "timt line" führt die Aktionen nicht automatisch aus! Du mußt die "time line" einen Objekt zuweisen und dann werden die Aktionen der "time line" für dieses Objekt ausgeführt. Dafür gibt es zwei besondere Aktionen; eine, um die "time line" für ein Objekt zu setzen und eine zweite, um den Zeitpunkt innerhalb der "time line" festzulegen. Wir verwenden wieder das Objekt `controller_enemy`. Es benötigt nur noch eine Aktion, das die entsprechende "time line" setzt, in seinem "creation event".

Das ist alles. Wie Du bemerken wirst, ist das Verwenden von "time lines" ein wenig mehr Arbeit, weil Du alle Flugzeuge, die auftauchen, bestimmen mußt aber es gibt Dir mehr Flexibilität und macht das Spiel viel interessanter.

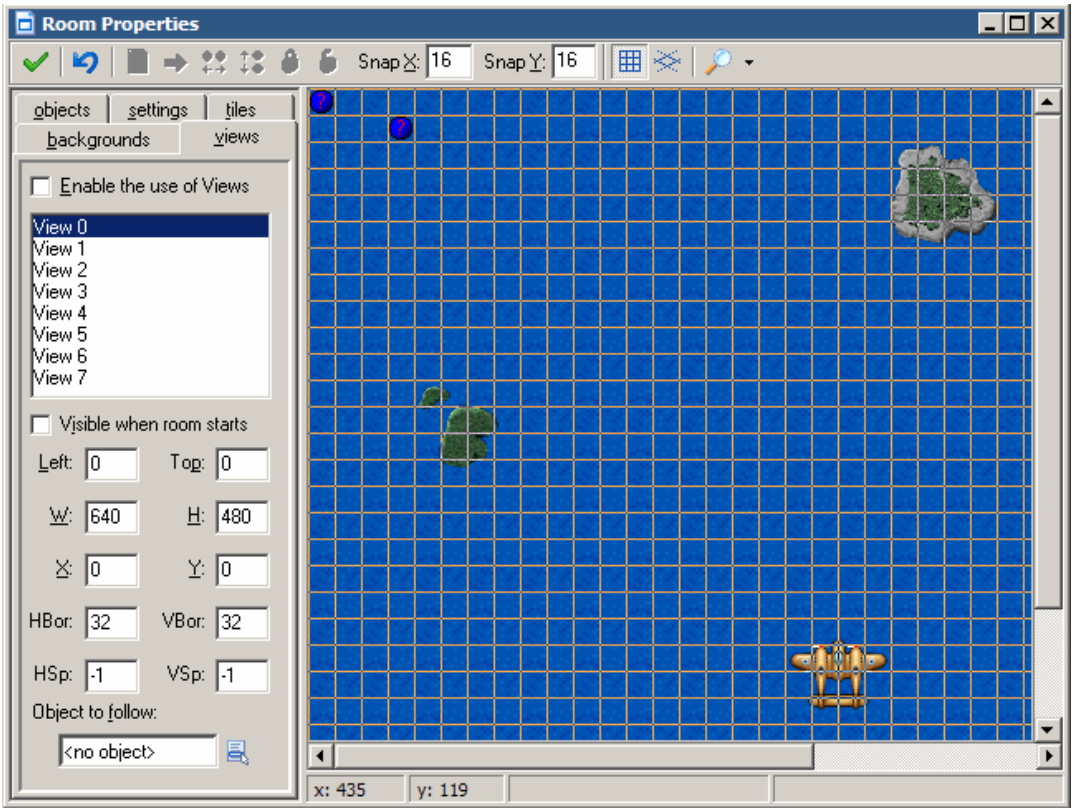
Es gibt viele Sachen, die Du mit "time lines" anstellen kannst. Hier haben wir sie verwendet, um den globalen Fluß des Spiels zu steuern aber Du kannst sie auch benutzen, um das Verhalten der Spielobjekte über die Zeit zu kontrollieren. Es gibt auch viele Tricks, die Du anwenden kannst. Zum Beispiel eine "time line" als Zyklus verwenden; setze am Ende einen Zeitpunkt, in dem Du die Position der "time line" wieder auf 0 setzt. Pausieren der "time line" für eine gewisse Zeit; füge einen Zeitpunkt hinzu, in dem Du eine Bedingung prüfst, wenn ja, setze die "time line" Position relative auf -1, das bedeutet, das im nächsten "step" der gleiche Zeitpunkt geschieht und die Bedingung wieder überprüft wird.
Somit schreitet die "time line" nicht weiter fort, solange die Bedingung nicht falsch wird. Instanzen können "time lines" auch verändern, basierend auf "events", usw.. Es sind sogar Variablen vorhanden, die die Ablaufgeschwindigkeit einer "time line" steuern. Somit stellen "time lines" eine mächtige Komponente dar.

Ein Raum mit einem "view" (Ansicht/Bildausschnitt)

Bislang täuschten wir den Spieler in der Art, daß wir nicht wirklich durch eine Spielwelt flogen, nur der Hitergrund bewegte sich (scrollte). Das birgt eine Zahl von Nachteilen. Im besonderen kann der Hintergrund nicht einfach wechsel (z. Bsp. ist es schwierig teilweise über Land zu fliegen). Auch ist es schwieriger verschiedene Aktionen schön über die Zeit verteilt geschehen zu lassen, obwohl man "time lines" verwendet.

In diesem abschließenden Abschnitt zeigen wir kurz einen anderen Weg auf einen "scrolling shooter" zu erstellen. Hier haben wir einen großen Raum durch den sich das Flugzeug wirklich bewegt. Wir verwenden einen sogenannten "view", sodaß der Spieler immer nur einen Teil des Raumes sieht. Um dies zu erreichen, müssen einige Änderungen am Spiel vorgenommen werden Wir diskutieren sie kurz hier.

Laß uns zuerst den zugrunde liegenden Raum erstellen. Wir geben dem Raum eine Breite von 640, wie vorher auch schon, aber diesmal setzen wir die Höhe mit 4800 an. Dem Hintergrundbild geben wir diesmal keine Eigenbewegung. Wir setzen Inseln an verschiedene Stellen in diesem Raum. Die Insel Objekte benötigen auch keine Eigenbewegung mehr und wir können auch das "step event" entfernen. Sie sind nun vollständig statische Objekte. Das Flugzeug platzieren wir an der Unterkante des Raumes und wir geben dem Flugzeug, in seinem "creation event" eine vertikale Geschwindigkeit von -2, um sicherzustellen, daß es mit einer gewissen Geschwindigkeit fliegt, wenn der Spieler nichts unternimmt. Schließlich definieren wir den "view". Klicke dafür auf die "views"-Registerkarte im "room"-Formular. Gib an, daß wir "views" ermöglichen wollen (enable). Definiere den ersten "**view**" und mach ihn sichtbar ab dem Start. Als "top position",
gib 4320, das heißt, der view befindet sich an der Unterkante des Raumes. Somit schaut es so aus:



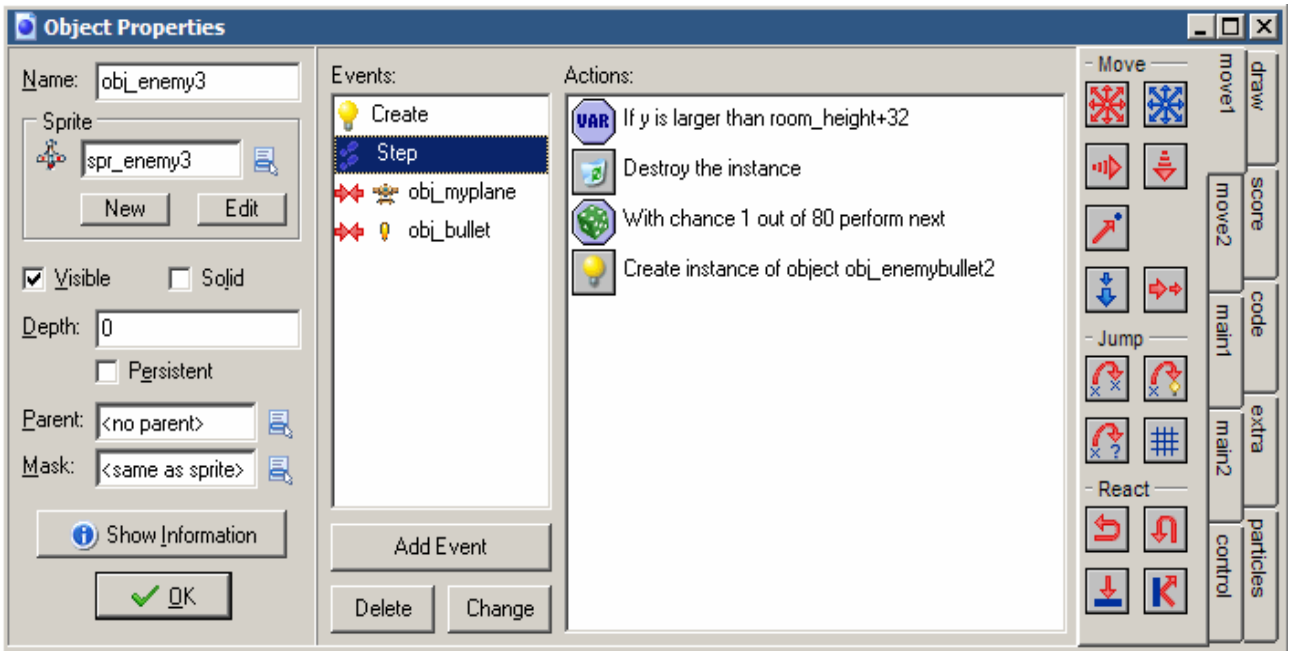
Wir müssen sicherstellen, das der "view" sich mit konstanter Geschwindigkeit bewegt. Dafür (miß) brauchen wir das 2step event" des life controller Objektes. In diesem nutzen wir die Aktion "set a variable" und "set variable view_top" relative auf -2. Variable `view_top` gibt die "top position" des ersten "views" an. (Sobald mehrere "views" vorhanden sind solltest Du "`view_top[0]`" benutzen.) Es gibt noch viele andere Variablen, die sich auf "views" beziehen aber wir brauchen sie hier nicht. Wir sollten auch überprüfen, ob `view_top` größer als 0 ist. Wenn es 0 wird, hat der "view" die Oberkante erreicht und wir sollten das Spiel anhalten.

Beachte, daß wir vermeiden wollen, daß der Spieler das Spielerflugzeug aus dem "view" herausbewegen kann. Deshalb prüfen wir, ob die Position des Spielerflugzeuges nicht zu hoch oder zu niedrig ist. Wir müssen jetzt die Variable `view_top` in diesen Vergleichen verwenden, um sicherzugehen, daß das Flugzeug innerhalb des "views" bleibt.

Wir haben ein ähnliches Problem mit dem Informationsanzeigefeld, der "health-bar", usw.. Wir haben sie an die Unterkante des Raumes gesetzt aber jetzt müssen wir sie an der richtigen Stelle relativ zu `view_top` setzen, um sicherzustellen, daß im Sichtfeld bleiben.

Bei den Kugeln, die das Flugzeug abfeuert, müssen wir die Geschwindigkeit etwas erhöhen, um den sich bewegenden "view" zu kompensieren. Auch sollten wir sie vernichten, wenn sie den view verlassen.

Es bleibt noch die Feindflugzeuge zu behandeln. Wir verwenden keine "time line" oder ein Kontroll-Objekt. Wir setzen die Flugzeuge bereits in den Raum aber wir lassen sie sich erst bewegen, wenn sie sichtbar werden. Somit setzen wir keine anfängliche Geshwindigkeit und im "speed event" prüfen wir erst, ob sie sich innerhalb des "views" befinden. Wenn ja lassen wir sie sich bewegen. Auch lassen wir sie nur schiessen, wenn sie innerhalb des "views" sind. Beispielsweise sieht für das dritte Feindflugzeug das "step event" wie folgt aus:



Beachte, daß wir eine kleinere vertikale Geschwindigkeit benutzen sollten, als vorher. Sowohl für die Feindflugzeuge, alsauch für die Kugeln sollten wir die Bedingungen ändern, wann sie zerstört werden sollen.

Das kompliziertesteste ist Feindflugzeug 4. Es muß von hinten ankommen. Der Trick geht wie folgt. Wieder platzieren wir es im Raum vom Start an, nur machen wir es unsichtbar. Zudem, solange es unsichtbar ist, lassen wir es nicht mit dem Spielerflugzeug oder Kugeln reagieren. Sobald es sich unterhalb des "views" befindet, machen wir es sichtbar und lassen es sich bewegen (mit höherer Geschwindigkeit). Schau Dir das Beispiel-Spiel an, um zu sehen, wie das erreicht wird.

Nun ist es an der Zeit den Raum zu erstellen. Wir platzieren Sätze von verschiedenen Feindflugzeugen an strategische Stellen innerhalb des Levels. Wieder ist es einfach Formationen und nette Herausforderungen zu erstellen. Dieses Spiel findest Du in der Datei `1945_6.gmd`. Wie Du siehst, kannst Du durch das Verwenden von "views" sehr interessante "scrolling shooter" ohne viel Mühe erstellen. Natürlich braucht das Spiel noch viel Arbeit. Es ist in einem gewissen Sinn noch zu einfach zu absolvieren und es ist definitiv zu kurz. Auch ein hübscher Endgegner zum Schluß wäre noch großartig. Verwende es einfach als Ansatz und mach da weiter.

Originaltext findet sich auf www.gamemaker.nl
Übersetzt mit Hilfe des Online Wörterbuches der TU-Chemnitz (http://dict.tu-chemnitz.de/dings.cgi)
(German translation by Augenzeuge. Arranged by Windapple)
Besucht die deutsche Game Maker Community auf www.gmaker.de !
This translation is dedicated to Mark Overmars.